UNITED STATES UTILITY PATENT APPLICATION

A PARSER FOR SIGNOMIAL AND GEOMETRIC PROGRAMS

PRIORITY IS CLAIMED TO PROVISIONAL APPLICATION
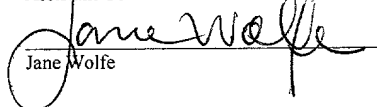SERIAL NO. 60/177,021, FILED JANUARY 19, 2000

Inventors:

Stephen Boyd
Xiling Shen
Mar Hershenson
Lieven Vandenberghe
Cesar Crusius
Dave Colleran
Sunderarjan Mohan

Attorney Docket No.
4363P001

Prepared by:
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard, 7th FL
Los Angeles, CA 90025-1026
408-720-8300

A PARSER FOR SIGNOMIAL AND GEOMETRIC PROGRAMS

PRIORITY IS CLAIMED TO PROVISIONAL APPLICATION
SERIAL NO. 60/177,021, FILED JANUARY 19, 2000

FIELD OF THE INVENTION

The invention relates to the field of mathematical optimization, particularly the

recognition and parsing of signomial and geometric programs such that they may be accepted

and solved by signomial and geometric program solvers.

BACKGROUND AND PRIOR ART

Geometric programming is a unique class of mathematical problems that is useful in the

study of optimization problems. The theory of Geometric Programming was initially developed

over three decades ago and culminated in a publication by R.J. Duffin, E.L. Peterson and C.M.

Zener (Geometric Programming, John Wiley & Sons, 1967). This publication describes a

Geometric Program (GP) as an optimization problem having an objective function and a set of

constraints which all must meet certain mathematical criteria. Perhaps the most important

property of GPs is that they can be solved, with great efficiently, and globally, using recently

developed interior-point methods.

Since the impact of GPs can be seen in fields of engineering design, manufacturing,

economics, statistics, and chemical equilibrium, it is apparent that the ability to solve GPs can

have many benefits. A very simple GP can be solved by hand, but more complex GPs

effectively must be solved with the aid of computer programs. For example, a very complex GP

that may be used to describe a circuit design problem may have hundreds of optimization

variables and thousands of constraints. The mathematical operations needed to solve a GP of

this magnitude would be too cumbersome to perform by hand; therefore, optimization programs are needed to solve most GPs.

Several optimization programs have been written that can solve GPs. For example, two popular GP solvers are MOSEK™ (product of EKA Consulting Aps, Vissenbjerg, Denmark) and COPL_GP (developed by members of the Computational Optimization Laboratory, Department of Management Science, University of Iowa). However, these programs require that GPs be parsed and represented in a particular compact numeric format that can be read and, if possible, globally solved by a solver program. Parsing a GP into this particular format typically requires extensive labor by one or more people. Moreover, if this compact numeric format were generated by hand, any changes to the functions, variables or coefficients contained in the GP would require extensive labor to modify the problem. Thus, it would be advantageous to have a parser program that can automatically convert a geometric program to this compact numeric format.

Software tools exist that will parse and solve a wide variety of optimization problems. Two well known examples of such tools are the software packages AMPL™ (product of the Bell Laboratories Division of Lucent Technologies Inc., Murray Hill, NJ) and GAMS™ (product of the GAMS Development Corporation, Washington, DC) which incorporate a variety of backend solvers. These packages allow a user to input one of a variety of optimization problems, in a standard algebraic format, and then will parse the problem into a format that is readable by a solver. However, these programs are very general in function and lack the sophistication required to identify and/or parse a GP. Some parsers have been developed to specifically parse specific classes of optimization problems. For example, a program called SDPSOL (developed by Dr. Stephen Boyd and members of his group at the Information Systems Laboratory,

Department of Electrical Engineering, Stanford University) is a parser/solver program specifically designed for semidefinite programming. Similarly, there is a need for a parser program that can identify and parse a GP because current optimization software tools lack the ability to accomplish these tasks.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments of the invention. In the drawings:

Figure 1 is a flow diagram showing the operations performed by certain exemplary embodiments of the invention.

Figure 2A is a first half of a flow diagram showing a more detailed look at the operations performed in block 130 of **Figure 1** according to one embodiment of the invention

Figure 2B is a second half of a flow diagram showing a more detailed look at the operations performed in block 130 of **Figure 1** according to one embodiment of the invention.

Figure 3 is a flow diagram showing a more detailed look at the operations performed in block 130 of **Figure 1** for an alternate embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, numerous specific details are set forth to provide a thorough understanding of the invention. However, it is understood that the invention may be practiced without these specific details. In other instances, well-known mathematical concepts, structures and techniques have not been shown in detail in order not to obscure the invention.

Background on Geometric Programming

In order to better understand the scope of the invention, it is important to introduce some mathematical terminology associated with geometric programming. For example, monomials, posynomials, and signomials are three types of mathematical functions found in geometric programs. These functions are defined below, beginning with a monomial.

Let $x_1,\ldots,x_2$ denote n real positive variables. A function $f: R^n \rightarrow R$ of the form

$$f(x) = cx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n},$$

where $c > 0$ and $a_i \in R$, is called a *monomial function*. Note that exponents can be any real numbers, including fractional or negative. We can refer to a monomial function more informally as just a *monomial* (of the variables $x_1,\ldots,x_n$). We refer to the constant c as the *coefficient* of the monomial. As an example, $2.3x_1^2x_2^{-0.15}$ is a monomial of the variables x_1 and x_2, with coefficient 2.3 and x_2-exponent -0.15.

Any positive constant is a monomial, as is any variable. Monomials are closed under multiplication and division: if f and g are both monomials then so are $f \times g$ and f / g. (This includes scaling by any positive constant.) A monomial raised to any power is also a monomial:

$$f(x)^\gamma = \left(cx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}\right)^\gamma = c^\gamma x_1^{\gamma a_1} x_2^{\gamma a_2} \cdots x_n^{\gamma a_n}.$$

In addition, a sum of zero or more monomials, *i.e.*, a function of the form

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_1 k} x_2^{a_2 k} \cdots x_n^{a_n k},$$

where $c_j \geq 0$, is called a *posynomial function* or, more simply, a *posynomial* (with K terms).

Any monomial is also a posynomial. Posynomials are closed under addition, multiplication, and nonnegative scaling. Posynomials can be divided by monomials (with the result also a posynomial): If f is a posynomial and g is a monomial, then f/g is a posynomial. If γ is a nonnegative integer and f is a posynomial, then f^γ always makes sense and is a posynomial (since it is the product of γ posynomials).

A generalization of a posynomial will also be encountered, in which the coefficients are allowed to be negative. A *signomial* is a function with the same form as a posynomial, as defined above, where the coefficients c_j are allowed to be negative. Signomials include as special cases posynomials (and therefore also monomials) and (positive or negative) constants. Signomials are closed under addition, subtraction, and multiplication. A signomial can be divided by a signomial that is nonzero and has only one term (*i.e.*, a monomial, or the negative of a monomial). A signomial raised to any nonnegative integer power is also a signomial.

Any signomial can be expressed as the difference of two posynomials, by collecting together the terms with positive coefficients, and also the terms with negative coefficients. We will use the notation f_+ to denote the *posynomial part* of the signomial f, and f_- to denote the *negative posynomial* part of f, so $f = f_+ - f_-$ where f_+ and f_- are both posynomials.

The following are some examples of monomials, posynomials and signomials. Suppose x, y, and z are (positive) variables. The functions

$$2x, \quad 0.23, \quad 2z\sqrt{x/y}, \quad 3x^2 y^{-.12} z$$

are monomials (hence, also posynomials and signomials). The functions

$$0, \quad 0.23 + x/y, \quad 2(1+xy)^3, \quad 2x + 3y + 2z$$

are posynomials (hence, also signomials) but *not* monomials. The functions

$$-2.2, \quad -x/y, \quad 2(1-xy)^3, \quad 2x + 3y - 2z$$

are signomials but *not* posynomials. Finally, the functions

$$(x + y)^{-1}, \quad 2(1 + xy)^{3.1}, \quad 1 + \log z$$

are *not* signomials.

Monomials, posynomials, and signomials are all functions that can be found in a geometric program ("GP"). A *standard form* GP is an optimization problem of the form show in Equation Set 1.

<div align="center">Equation Set 1</div>

$$
\begin{aligned}
\text{minimize} \quad & f_0(x) \\
\text{subject to} \quad & f_i(x) \le 1, \quad && i = 1, \ldots, m, \\
& h_i(x) = 1, \quad && i = 1, \ldots, p
\end{aligned}
$$

In Equation Set 1 f_i are posynomial functions, h_i are monomials, and x_i are the optimization variables. (There is an implicit constraint that the variables are positive, i.e., $x_i > 0$). In a GP in standard form the objective f_0 must be posynomial and it must be minimized; in addition, the equality constraints can only have the form of a monomial equal to one, and the inequality constraints can only have the form of a posynomial less than or equal to one. A solution to a GP is found when a minimum or maximum value of the objective is calculated from values of the optimization variables meeting the conditions set by the constraints. What makes the use of GPs powerful is that they can be solved globally, with great efficiency.

A generalization of a geometric program is a *signomial program* ("SGP"), which has the form shown in Equation Set 2.

<div align="center">Equation Set 2</div>

$$
\begin{aligned}
\text{minimize} \quad & f(x) \\
\text{subject to} \quad & a_i(x) \le b_i(x), \quad && i = 1, \ldots, m, \\
& c_i(x) = d_i(x), \quad && i = 1, \ldots, p
\end{aligned}
$$

In Equation Set 2 a_i, b_i and d_i are all *signomial* functions, and x_i are the optimization variables. (As in GP, there is an implicit constraint that the variables are positive, i.e., $x_i > 0$.) It is also required that the objective f be a posynomial.

Note the differences between the standard form GP (Equation Set 1) and the SGP (Equation Set 2):

- In an SGP, the constraint functions can be arbitrary signomials; in a GP they must be posynomials and monomials, respectively.

- In an SGP, the righthand side of the inequalities and equalities can also be signomials; in a GP, the righthand side of the inequalities and equalities is one.

From a computational point of view, there is a significant difference between a GP and an SGP. While the *globally optimal* solution of a GP can always be found efficiently (e.g., an absolute maximum, only a *locally optimal* solution of an SGP can be computed efficiently. Although it is possible to compute the globally optimal solution of an SGP, this can require prohibitive computation, even for relatively small problems.

Furthermore, it can be seen that a standard form GP with posynomial objective function f_0, posynomial constraint functions $f_1, \ldots, f_m$, and monomial equality functions $g_1, \ldots, g_p$, is already in SGP form, with $b_i = c_i = 1$ (which are signomials). Thus any standard form GP is also an SGP.

For several purposes of the invention it is convenient to transform a SGP into a special form in which the left and right-hand sides of each constraint are posynomials. This can be done by first expressing the inequality $a_i \leq b_i$ in Equation Set 2 as $a_i - b_i \leq 0$, and then splitting it into its posynomial and negative posynomial parts, and expressing it as $(a_i - b_i)_+ \leq (a_i - b_i)_-$. The

same can be done to each equality constraint, and examples of splitting a signomial into its

posynomial and negative posynomial are shown in Examples 1 and 2.

Example 1:　　$(2x + 3y - 2z)_+ = 2x + 3y,$　　$(2x + 3y - 2z)_- = 2z$

Example 2:　　$(2(1 - xy)^3)_+ = 2 + 6x^2y^2,$　　$(2 (1 - xy)^3)_- = 6xy + 2x^3y^3$

Ultimately, the SGP in Equation Set 2 can be rewritten as shown in Equation Set 3 below.

Equation Set 3

minimize　　$f(x)$

subject to　　$(a_i(x) - b_i(x))_+ \quad (a_i(x) - b_i(x))_- \quad i = 1,...,m$

　　　　　　　$(c_i(x) - d_i(x))_+ = (c_i(x) - d_i(x))_- \quad i = 1,...,p$

In this form of the SGP the objective and each of the left and right-hand sides of each constraint

is a posynomial function. For this reason we refer to the program in Equation Set 3 as the

posynomial form of the SGP.

One Embodiment of the Invention

Figure 1 illustrates the operations that are performed by certain exemplary embodiments

of the invention, hereinafter "the Parser," in order to parse an optimization problem such as a

signomial or geometric program. These operations are contained in box 100. First a user 110

must input a source file 115 that describes an optimization problem (e.g., an SGP), as shown in

block 120. This process of inputting source file 115 can occur via a user interface which may

include a computer operating system, a web browser, or other user-computer interfaces. Each

line of source file 115 will include one of the following: 1) an optimization variable declaration,

2) an internal variable assignment or 3) an algebraic expression. In addition, the algebraic

expressions contained in the source file will collectively comprise: 1) an objective function, 2) one or more equality constraints and 3) one or more inequality constraints.

A very simple example of an optimization program, in this case a SGP, is shown in Equation Set 4:

<div align="center">

Equation Set 4

</div>

$$\text{minimize} \quad x_1 x_3 + x_1 x_2 / x_3$$
$$\text{subject to} \quad x_1 x_3^{-0.3} + 1 \quad x_3 - x_1 x_2 / x_3 \, x_2^{1.2},$$
$$x_3 \quad 30,$$
$$x_1 - 4.1 \, / x_2 \, x_3 = 0,$$

This problem can be specified in the Parser language as shown in Equation Set 5:

<div align="center">

Equation Set 5

</div>

```
variable x1, x2, x3;
minimize x1*x3+x1*x2/x3;
x1 *2^-0.3 + 1 < x3 – x2^1.2;
x2 < 30;
x1 – 4.1 sqrt (x2/x3) == 0;
```

There is a very close similarity between this the Parser source file (i.e., Equation Set 5) and the mathematical description shown in Equation Set 4. In the first line, x1, x2 and x3 are declared to be optimization variables, and in the second line the objective is defined by an algebraic expression that includes a set of mathematical terms. For the purposes of the invention an algebraic expression is defined as a set of one or more variables and, in most problems, constants joined together by mathematical operators and functions including by not limited to addition, subtraction, multiplication and division. In addition, a mathematical term is defined as a subset of an algebraic expression that can independently form a mathematical function. Lines three and four of Equation Set 5 specify inequality constraints, and the last line specifies an

equality constraint. Constraints are also defined by algebraic expressions which each include a set of mathematical terms.

At block 130 the Parser will determine if the algebraic expressions contained in the source file 115 form a GP. This provides the advantage of knowing that a *global* (e.g., an absolute minimum) solution to the optimization problem will be found as opposed to a *local* (e.g., a relative minimum) solution. Block 130 is illustrated in more detail in **figures 2A and 2B** and will soon be discussed.

If block 130 is successful then at block 150 the Parser will continue to parse the GP by converting it from its algebraic format (e.g., Equation Set 5) to a compact numeric format ("CNF"), thus creating CNF file 175. Otherwise, if block 130 is unsuccessful then an error message will be reported to user 110, indicating the type of error and at which line of source file 115 the error occurred. Types of errors include but are not limited to syntax and mathematical infeasibility. It will be understood that errors may be immediately reported to user 110, or the Parser can wait until all the lines of source file 115 are read and report the total number of errors found and their locations in source file 115. If an error message is reported then user 110 may make corrections and re-execute the Parser program.

Alternatively, the Parser may report the error stating that the optimization problem cannot be reduced to a GP in standard form, but that it still meets the mathematical criteria of a SGP. In this case user 110 will have the option to successfully re-execute the Parser program in order to create CNF file 175 and have an external SGP solver attempt to find a local solution. However, it would be advantageous for user 110 to modify source file 115 such that the optimization problem may be reduced and parsed as a GP instead of SGP. This modification will provide a GP solver the opportunity to find a global solution to the optimization problem.

The conversion that occurs at block 150 is accomplished when the Parser successfully coverts a GP or SGP from its algebraic format to a CNF. The CNF is a format that is specific to the type of solver program that it is being using in conjunction with the Parser. The CNF is comprised of a set of matrices that represent the variables, coefficients, and exponents of the algebraic equations that form a GP or SPG. These matrices are created when the Parser converts each individual algebraic equation, as expressed in the GP or SGP, into a set of numeric values that represent the equations' optimization variables by their types, coefficients, and exponents. These numeric values are further arranged into a group of matrices that are stored in CNF file 175. These matrices are stored in such a manner that a solver program will be capable of reading CNF file 175 and recognizing a specific SGP or GP that it will attempt to solve.

Once CNF file 175 is created, the Parser will route it to solver 160. Solver 160 is capable of globally solving GPs and can process CNF file 175. For example, solvers MOSEK™ and COPL_GP are will know in the art and will accept CNF file 175 and are capable of solving both SGPs and GPs using interior point methods. All GP solvers require that a GP be inputted in a unique format; hence CNF file 175 produced by the Parser. Currently, without the aid of the Parser, this unique format file must be generated by hand. It can be appreciated that the Parser can automatically generate CNF file 175 as described earlier.

Figures 2A and 2B show a more detailed look at the operations included in block 130 of **figure 1** according to one embodiment of the invention. From block 120 the Parser begins processing source file 115 to find the algebraic expressions, as shown is block 200. If a line contains a variable declaration, then the Parser stores the variables into memory, or if a line contains an internal variable assignment, then the Parser stores the internal variable and its assigned expression into memory. From block 200 the Parser will read a new algebraic

expression at block 205. At block 210 the Parser recognizes, in the algebraic expression, any internal variables representing a previously assigned expression and, if necessary, makes one or more substitutions. The Parser will then advance to block 220 where it attempts to convert all the mathematical terms in the algebraic expression to signomial functions, thus creating a signomial expression. This conversion, if successful, allows for the performing of block 250 with great efficiency as described later.

If block 220 is successful, as determined by block 230, then the Parser will advance to block 240. Otherwise, if block 220 is unsuccessful then an error message will be reported to user 110, indicating the type of error and at which line of source file 115 the error occurred. Types of errors include but are not limited to syntax and mathematical infeasibility. It is understood that errors may be immediately reported to user 110, or the Parser can wait until all the lines of source file 115 are read and report the total number of errors found and their locations in the 115. If an error message is reported then user 110 may make corrections and re-execute the Parser program.

At block 240 the Parser will attempt to simplify the signomial expression created in block 220. For example, simplification can occur when a combination of two identical signomial functions of opposite signs are mathematically cancelled. The Parser will then advanced to block 250 where it will attempt to reduce the signomial expression to one of the following: 1) a posynomial objective, 2) a posynomial inequality, or 3) a monomial equality. Specifically, a posynomial inequality shall be a posynomial function being less than or equal to one, and a monomial equality shall be monomial function being equal to one. The fact that block 220 generated a signomial expression allows the Parser to apply a test that will determine if the reduction of the constraints in block 250 is successful.

Not all signomial constraints can be reduced to either a posynomial or monomial, but after block 220 it is straightforward to make this determination at block 250. At block 250 the Parser will first rewrite the signomial expression in its posynomial form. This process is shown in Equation Set 3 and can be applied to either an inequality or an equality, for example,

$$(a_i(x) - b_i(x))_+ \quad (a_i(x) - b_i(x))_- \quad i = 1,...,m,$$

or
$$(c_i(x) - d_i(x))_+ = (c_i(x) - d_i(x))_- \quad i = 1,...,p.$$

The Parser will then determine if the right-hand side of the posynomial form constraint, i.e.,

$$(a_i(x) - b_i(x))_- , \quad i = 1,...,m \quad \text{or} \quad (c_i(x) - d_i(x))_- , \quad i = 1,...,m$$

is a monomial. In addition, in the case of the equality the Parser will also determine if the left-hand side is a monomial. Ultimately, if block 250 is successful a constraint can be expressed as either

$$(a_i(x) - b_i(x))_+ \,/\, (a_i(x) - b_i(x))_- \quad 1 \quad i = 1,...,m,$$

or
$$(c_i(x) - d_i(x))_+ \,/\, (c_i(x) - d_i(x))_- = 1 \quad i = 1,...,p.$$

This notation reflects the notation shown in Equation Set 1, which represents a standard form GP. Therefore, with some abuse of notation, any SGP which can be transformed into a standard form GP in this way can be referred to as a *geometric program.*

As a simple example, consider the SGP:

$$\begin{aligned}
\text{minimize} \quad & x_1 x_3 + x_1 x_2 / x_3 \\
\text{subject to} \quad & x_1 x_2^{-0.3} + 1 \quad x_3 - x_2^{1.2}, \\
& x_2 \le 30, \\
& x_1 - 4.1 \sqrt{x_2 / x_3} = 0,
\end{aligned}$$

which evidently is not a standard form GP. Therefore, the SGP is put in posynomial SGP form:

$$\begin{aligned}
\text{minimize} \quad & x_1 x_3 + x_1 x_2 / x_3 \\
\text{subject to} \quad & x_1 x_2^{-0.3} + 1 + x_2^{1.2} \le x_3,
\end{aligned}$$

$$x_2 \quad 30,$$
$$x_1 = 4.1 \sqrt{x_2 / x_3} \ .$$

Since each right-hand side is a monomial, we can transform the problem to a standard form GP by dividing by the right-hand sides:

minimize $x_1 x_3 + x_1 x_2 / x_3$

subject to $x_1 x_2^{-0.3} x_3^{-1} + x_3^{-1} + x_2^{1\,2} x_3^{-1} + x_2^{1\,2} x_3^{-1} \quad 1,$

$(1/30) x_2 \quad 1,$

$(1/4.1) x_1 x_2^{-0.5} x_3^{0.5} = 1.$

To continue the example, let us consider a closely related problem, with the sign of the constant 1 appearing in the first inequality constraint changed:

minimize $x_1 x_3 + x_1 x_2 / x_3$

subject to $x_1 x_2^{-0.3} -1 \quad x_3 - x_2^{1.2},$

$x_2 \quad 30,$

$x_1 - 4.1 \sqrt{x_2 / x_3} = 0.$

This problem cannot be transformed to a standard form GP, since the right-hand side of the first inequality constraint, in posynomial SGP form, now has two monomial terms. The practical consequence is that the ability to solve the problem globally is lost; only a local minimum can be found.

It is also understood that the Parser can parse signomial and geometric programs having objectives that are either minimized or maximized. In most cases and in the examples provided herein, objectives are minimized. However, a user may request that the Parser maximize the objective, and, therefore, the objective may be maximized by the Parser by finding the minimum of it's inverse. For example, the minimum of objective X can be expressed by the Parser as the

maximum of the objective 1/X; and similarly, the maximum of objective Y can be expressed by the parser as the minimum of objective 1/Y.

Ultimately, if block 250 is successful, as determined by block 260, and all lines of source file 115 have been read, as determined by block 270, then the Parser will advance to block 150 of **figure 1**. If the last line has not been read, as determined by block 270, the Parser will repeat the flows of **figures 2A and 2B** until all the lines of source file 115 have been read and no errors have been encountered. Once the Parser has reached block 150 it is clear that optimization problem inputted in block 120 is in fact a GP. Subsequently, at block 150, the Parser further converts the GP to a CNF, thus converting source file 115 to CNF file 155. CNF file 155 can be inputted to solver 160, as discussed earlier, and solver 160 will attempt to find a global solution to the GP.

If solver 160 calculates a solution to the GP represented by CNF file 155, then it creates solution file 165, containing a numeric solution to the objective and corresponding numeric solutions to the optimization variables. Otherwise, an error will be reported in solution file 165. Solution file 165 is routed back to the Parser where it is translated into a user-readable format (block 170) and expressed in output file 175. An example of output file 175 is shown in Example 3 which displays the global minimum solution to the objective function and the corresponding values of the optimization variables (x_1, x_2 and x_3), as declared in source file 115.

<div align="center">

Example 3

</div>

```
problem=GP
status=optimal
obj_value=1606.14
x1=29.999680001707
x2=37.433796562618
x3=0.69919504957707
```

Alternate Embodiment of the Invention

One embodiment of the invention exists which implements the identical blocks shown in

figure 1 with the exception of block 130. **Figure 3** shows a more detailed look at the operations

included in block 130 for an alternate embodiment. From block 120 the Parser begins processing

source file 115 to locate algebraic expressions, as shown is block 300. If a line contains a

variable declaration, then the Parser stores the variables into memory, or if a line contains an

internal variable assignment, then GBLAB stores the internal variable and its assigned

expression into memory. From block 300 the Parser will read a new algebraic expression at

block 310. At block 320 the Parser recognizes, in the algebraic expression, any internal variables

representing a previously assigned expression and, if necessary, makes one or more substitutions.

The Parser will then advance to block 330 where it attempts to combine the mathematical terms

such that the algebraic expression reduces to one of the following: 1) a posynomial objective, 2)

a posynomial inequality, or 3) a monomial equality. Specifically, a posynomial inequality shall

be a posynomial function being less than or equal to one, and a monomial equality shall be

monomial function being equal to one.

Combining the mathematical terms at block 330 shall first involve the Parser identifying

each mathematical term as a signomial, posynomial or monomial. Then the Parser will

determine if the operators and functions that relate the mathematical terms allow the algebraic

expression to be reduced as described above. This process is more intensive than that of the

preferred embodiment because the mathematical terms are not all treated as signomials. Rather,

the mathematical terms are treated as separate classes of terms (e.g., posynomials and

monomials); therefore, the set of mathematical rules that apply to combining the terms and

reducing the algebraic expression is more expansive. Moreover, when compared to the

embodiment described with reference to Figures 2A-B, the cancellation of mathematical terms will not be as obvious to the Parser since the terms are not all treated as signomials.

Ultimately, if block 330 is successful, as determined by block 340, and all lines of source file 115 have been read, as determined by block 350, then the Parser will advance to block 150 of **figure 1**. If the last line has not been processed, as determined by block 350, the Parser will repeat the blocks of **figure 3** until all the lines of source file 115 have been processed and no errors have been encountered. Once block 150 is reached, the Parser will proceed through the final blocks of **figure 1** in a manner identical to that of the embodiment of Figure 2A-B. Once the Parser has reached block 150 it is clear that optimization problem inputted in block 120 is in fact a GP.

Otherwise, if block 330 is unsuccessful then an error message will be reported to user 110, indicating the type of error and at which line of source file 115 the error occurred. Types of errors include but are not limited to syntax and mathematical infeasibility. It will be understood that errors may be immediately reported to user 110, or the Parser can wait until all the lines of source file 115 are read and report the total number of errors found and their locations in the 115. If an error message is reported then user 110 may make corrections and re-execute the Parser program.

Alternatively, the Parser may report the error stating that the optimization problem cannot be reduced to a GP in standard form, but that it still meets the mathematical criteria of a SGP. In this case user 110 will have the option to successfully re-execute the Parser program in order to create CNF file 175 and have an external SGP solver attempt to find a local solution. However, it would be advantageous for user 110 to modify source file 115 such that the optimization

problem may be reduced and parsed as a GP instead of SGP. This modification will provide a GP solver the opportunity to find a global solution to the optimization problem.

While the invention has been described in terms of several embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The method and apparatus of the invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting on the invention.